Feature Engineering for Machine learning in Graphs

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

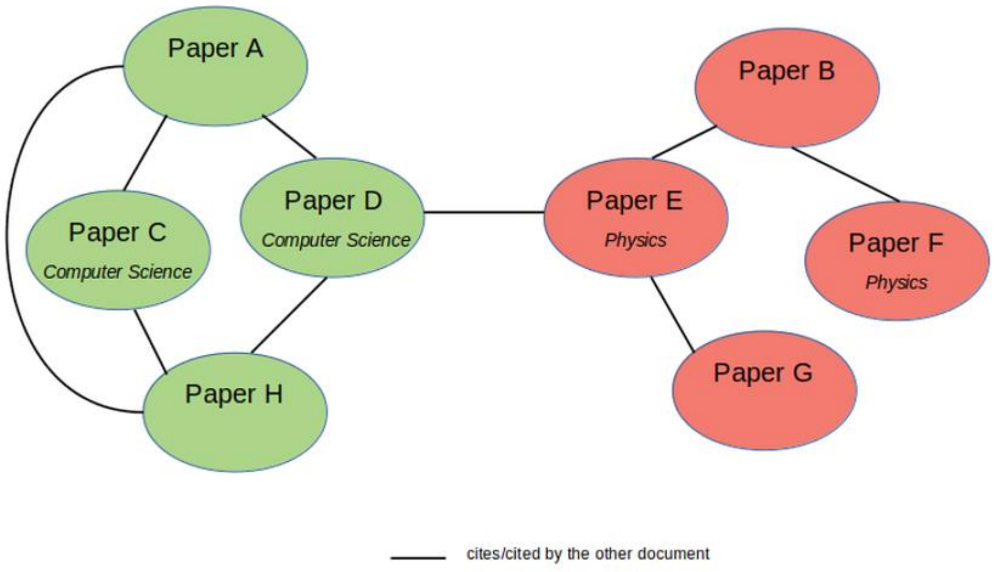
네트워크 과학 연구실 NETWORK SCIENCE LAB

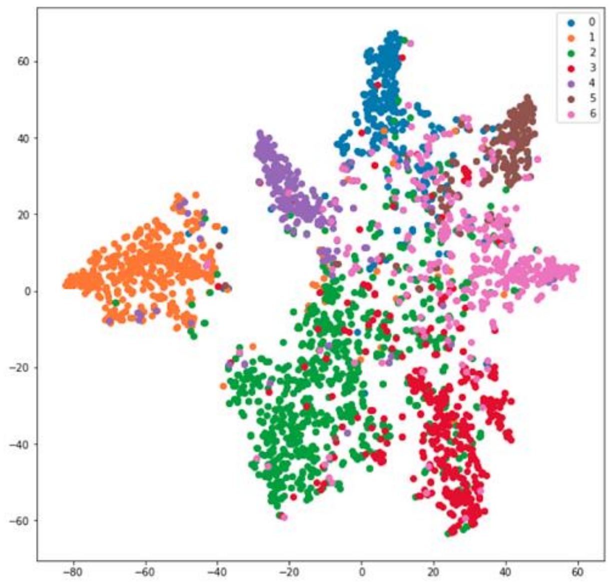
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

Contents

Node classification task

- Predicting the classes or labels of nodes.
- For example, detecting the paper field in a citation network.

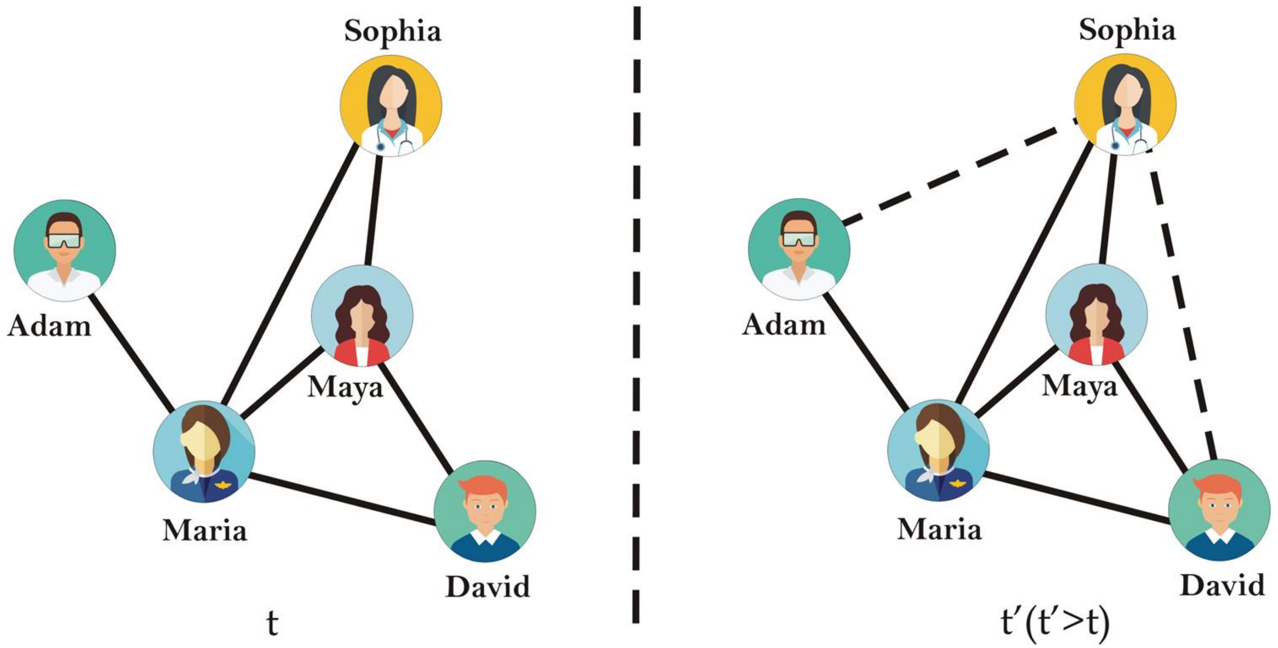




네트워크 과학 연구실 NETWORK SCIENCE LAB
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

Link prediction task

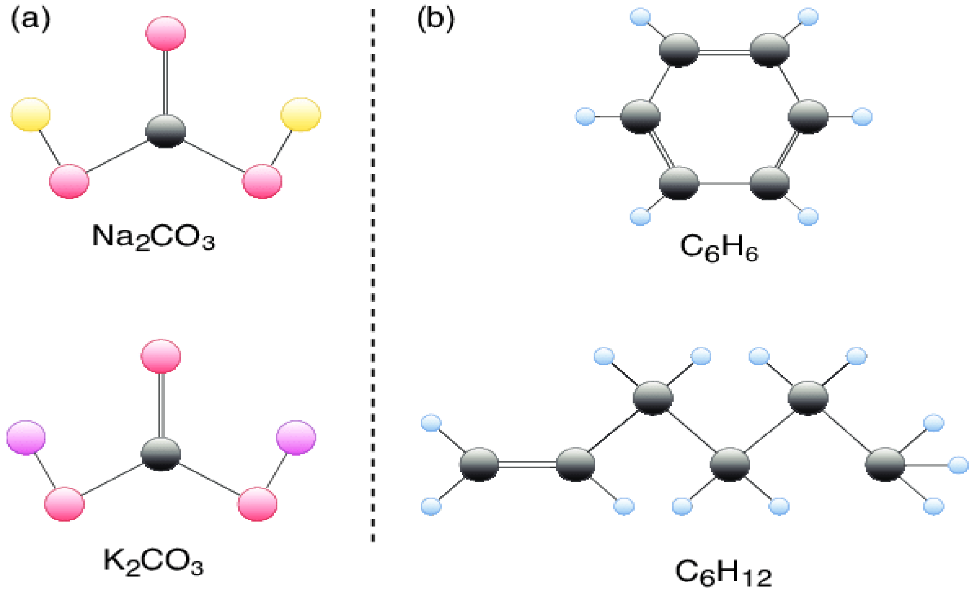
- Predicting the classes or labels of nodes.
- For example, a social networking service suggests possible friend connections based on network data.



https://www.nature.com/articles/s41598-019-57304-y

Graph classification task

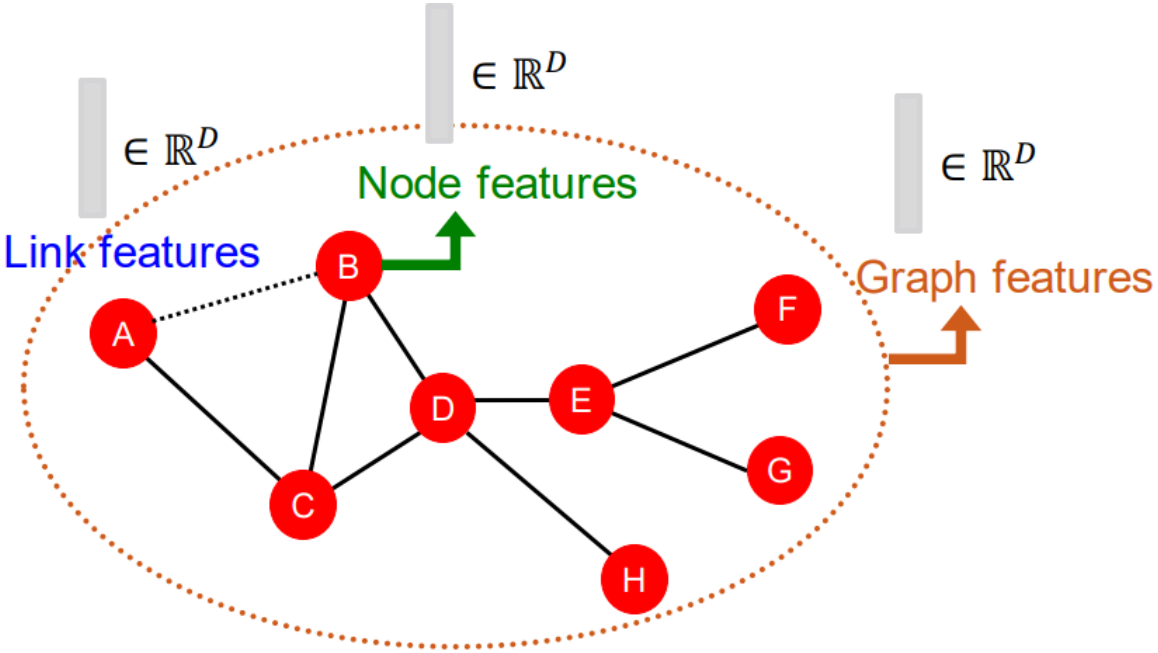
- Classifying a graph itself into different categories.
- Inputs: A collection of graphs.
- For example, determining if a chemical compound is toxic or non-toxic by looking at its graph structure.



https://www.researchgate.net/figure/An-illustration-of-molecule-graph_fig1_349880436

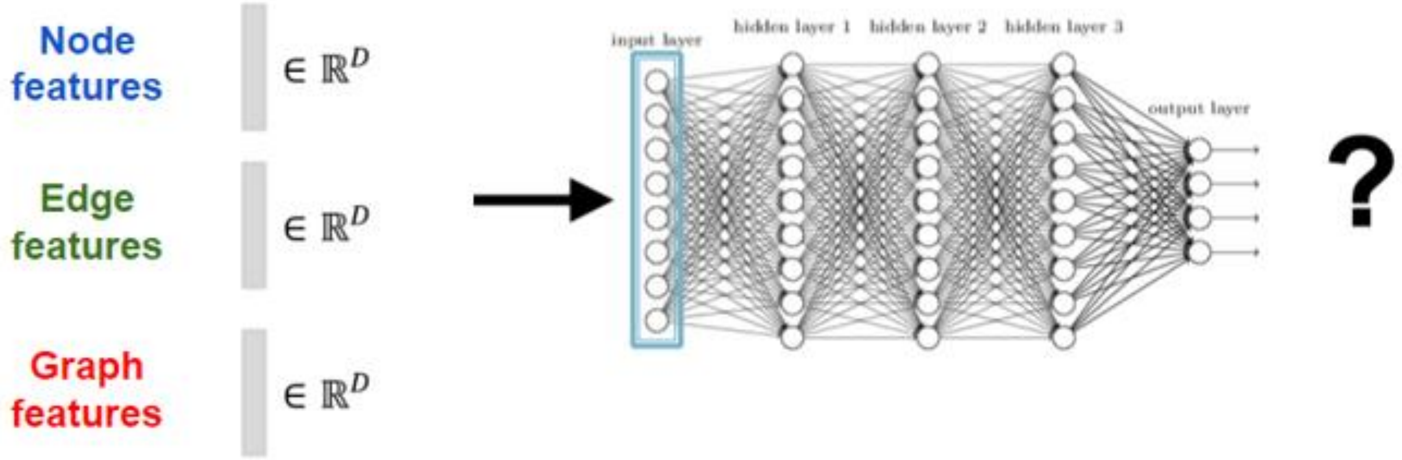
Traditional Machine Learning pipeline

- ➢ Design features for nodes/links/graphs
- ➢ Obtain features for all training data

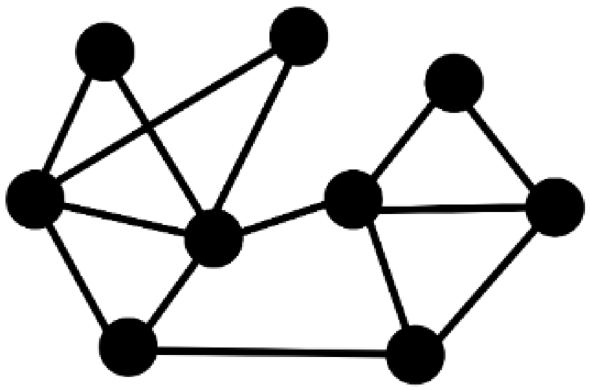


Traditional Machine Learning pipeline

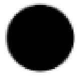
- **Feature extraction**:
  - Node features
  - Edge features
  - Graph features
- **Train a ML model**:
  - Logistic Regression
  - Random forest
  - Neural network, etc.
- **Apply the model:**
  - Given a new node/link/graph, obtain its features and make a prediction



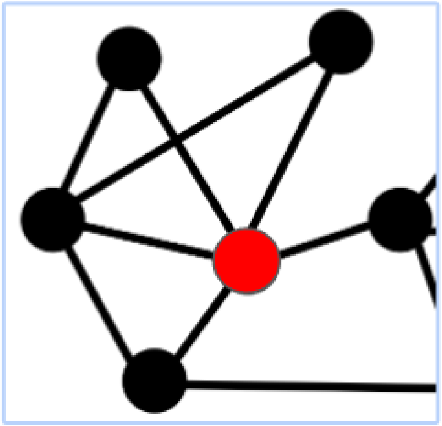
- Using effective features over graphs is the key to achieving good model performance.
- Besides the original node/edge/graph features, can we embed topology structure into node/edge/graph features?



Graph



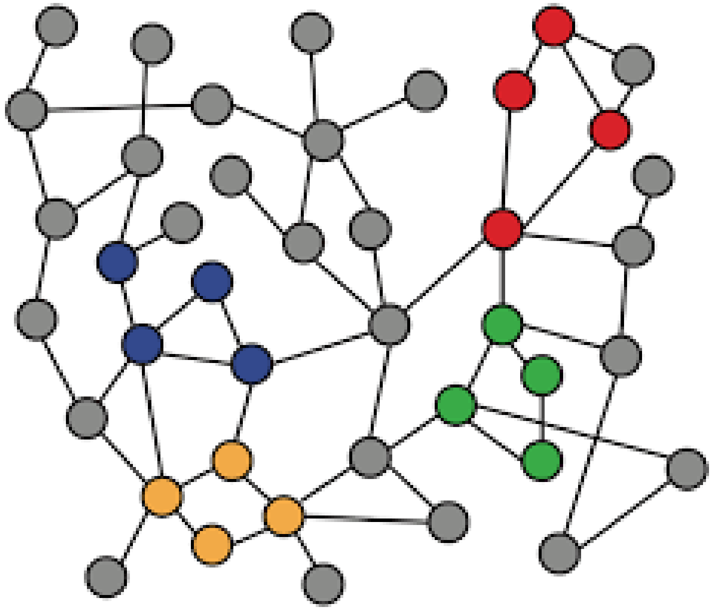
Node prediction



With neighboring topology

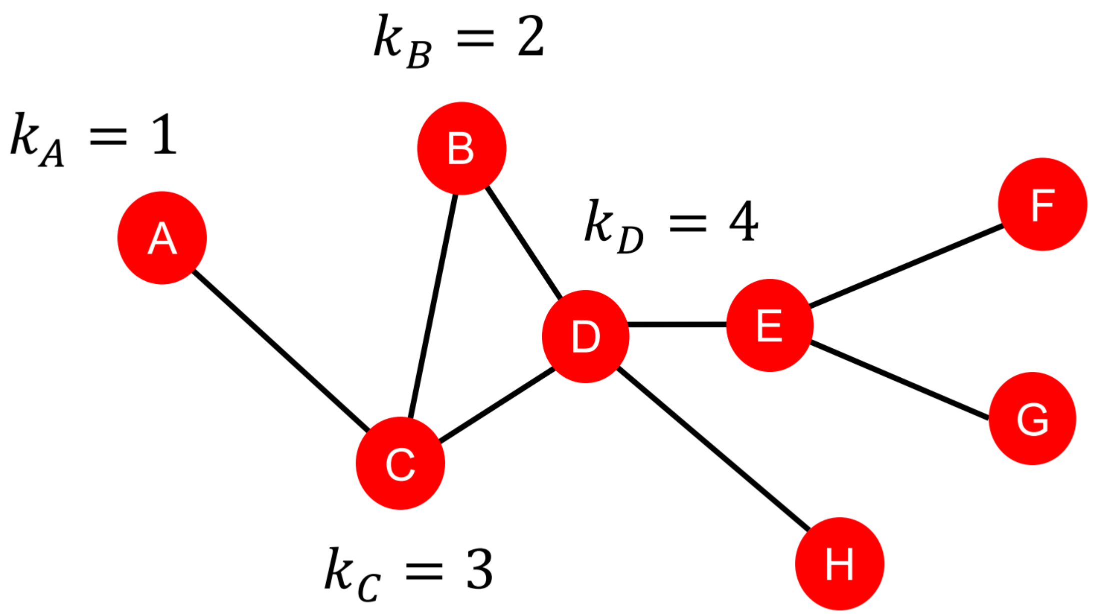
- Goal:

  Characterize the structure and position of a node in the network:
- Importance-based features
  - Node degree
  - Node centrality
- Structure-based features
  - Node degree
  - Clustering coefficient
  - Graphlets



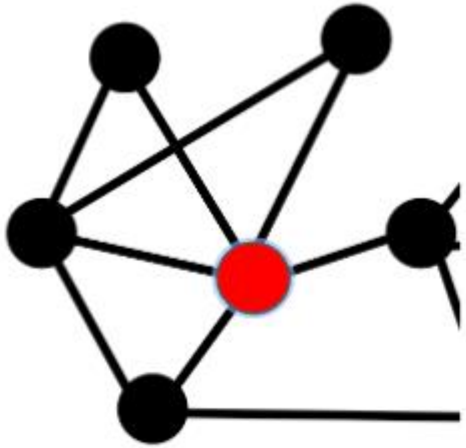
Node Features: Degree

- The degree k_v of node v is the number of edges (neighboring nodes) the node has.
- Treats all neighboring nodes equally.



Node Features: Node Centrality

- Node degree counts the neighboring nodes without capturing their importance.
- Node centrality cv takes the node importance in a graph into account.
- Different ways to evaluate importance:
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - and many others….



- **Eigenvector Centrality**
  - A node v is important if surrounded by important neighboring nodes u ∈ N(v).
  - We model the centrality of node v as the sum of the centrality of neighboring nodes recursively:

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad \Longleftrightarrow \quad \lambda \boldsymbol{c} = \boldsymbol{A}\boldsymbol{c}$$

- $\boldsymbol{A}$: Adjacency matrix
  $\boldsymbol{A}_{uv} = 1$ if $u \in N(v)$
- $\boldsymbol{c}$: Centrality vector
- λ: Eigenvalue

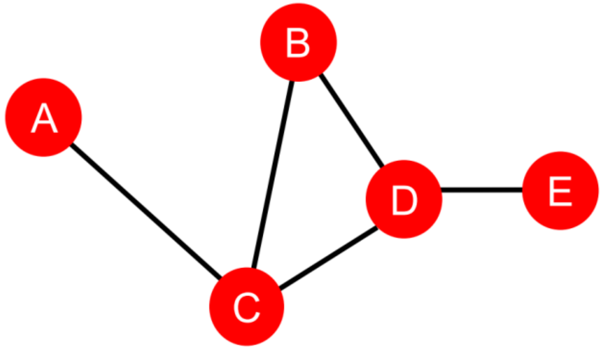
  - We can see that centrality ***c*** is the eigenvector fo ***A***

Node Features: Node Centrality (2)

- Betweenness Centrality
  - A node is important if it **lies on many shortest paths** between other nodes.

$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths betwen } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

  - For example:

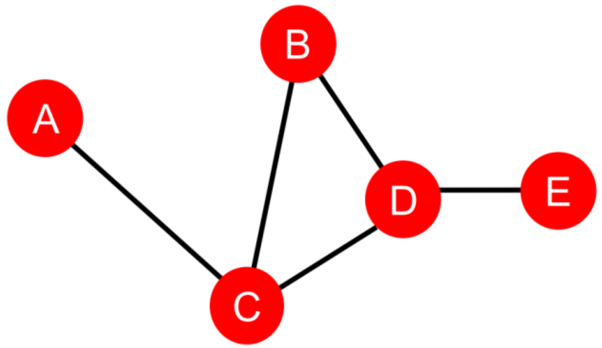


- C_A = C_B = C_E = 0
- C_C = 3 (A-C-B, A-C-D, A-C-D-E)
- C_D = 3 (A-C-D-E, B-D-E, C-D-E)

- Closeness Centrality
  - A node is important if it has **small shortest path lengths** to all other nodes.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

  - For example:



- C_A = 1/(2 + 1 + 2 + 3) = 1/8
(A-C-B, A-C, A-C-D, A-C-D-E)
- C_D = 1/(2 + 1 + 1 + 1) = 1/5
(D-C-A, D-B, D-C, D-E)

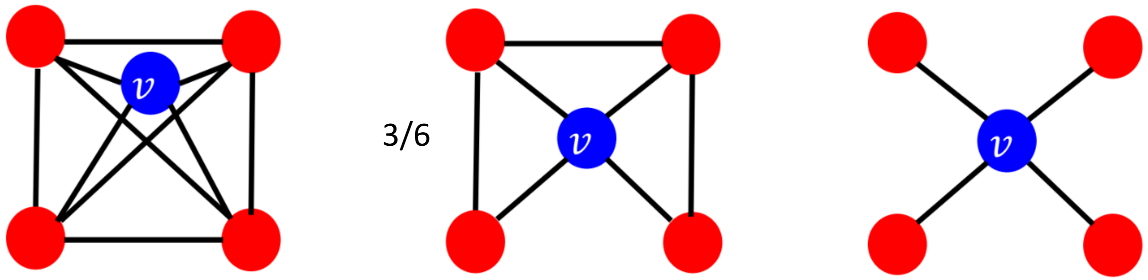
- Measures **how connected v's neighboring nodes** are:

$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\boxed{\binom{k_v}{2}}} \in [0,1]$$

#(node pairs among k_v neighboring nodes)
In our examples below the denominator is 6 (4 choose 2).

- For example:

6/6 3/6 0/6

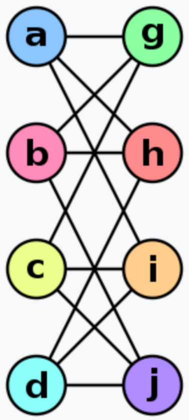
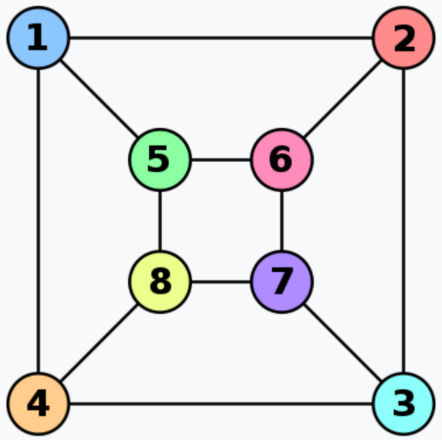


$e_v = 1$ $e_v = 0.5$ $e_v = 0$

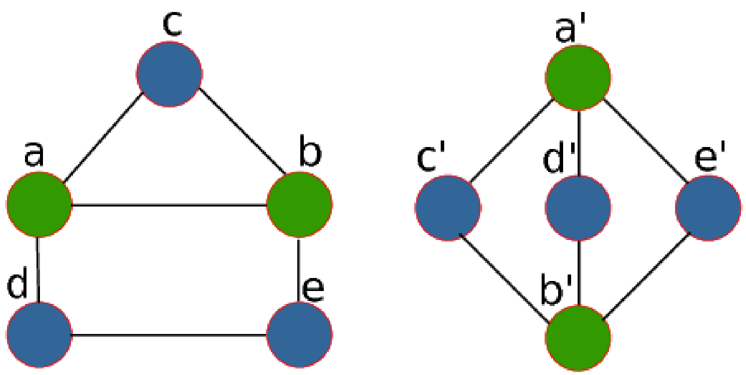
네트워크 과학 연구실 NETWORK SCIENCE LAB 가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

Graph Isomorphism

- **Isomorphic graphs** having the same number of vertices, edges, and also the same edge connectivity.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

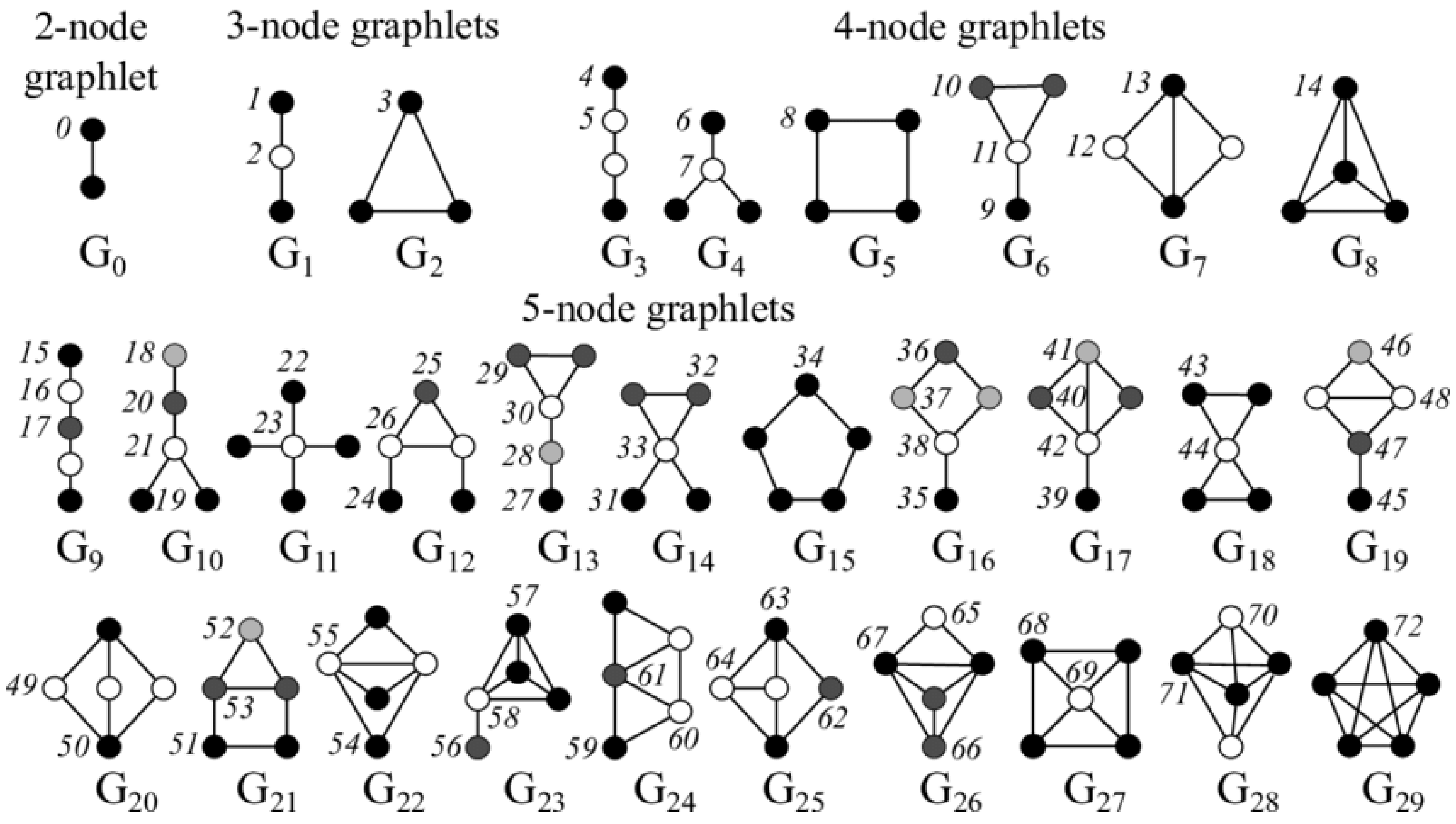
Isomorphic



Non-Isomorphic

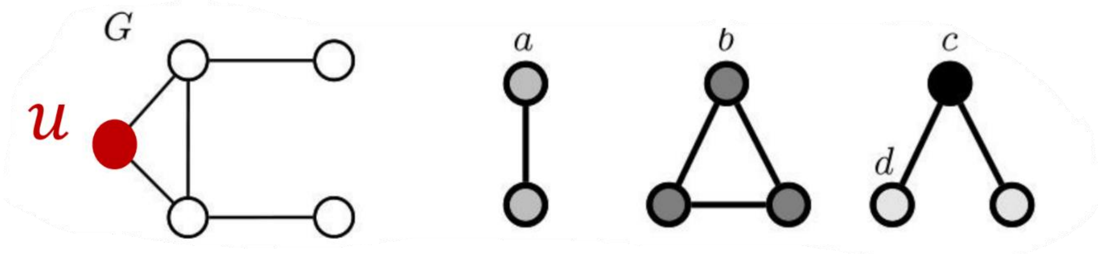
네트워크 과학 연구실 NETWORK SCIENCE LAB
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

- Graphlets are **induced, non-isomorphic** subgraphs that **describe the structure of node u's network neighborhood.**

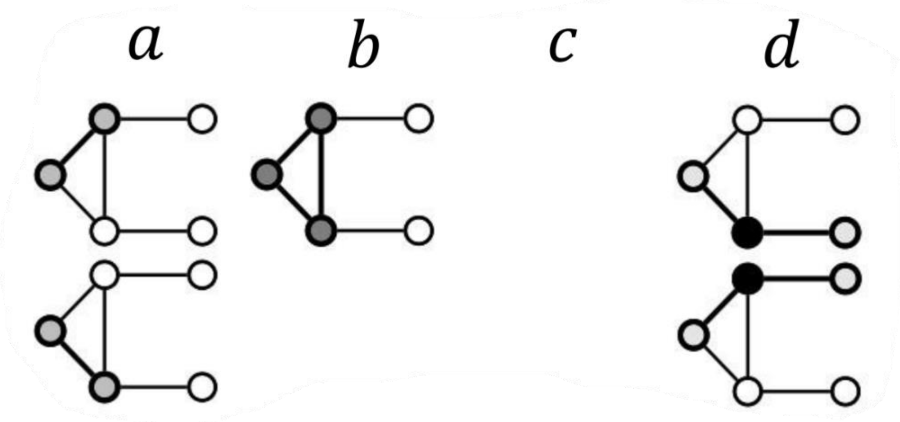


Node Features: Graphlet

- Graphlet Degree Vector (GDV): A count vector of graphlets rooted at a given node.
- Graphlet degree vector provides a measure of a node's local network topology (more detail than node degrees or clustering coefficient.)



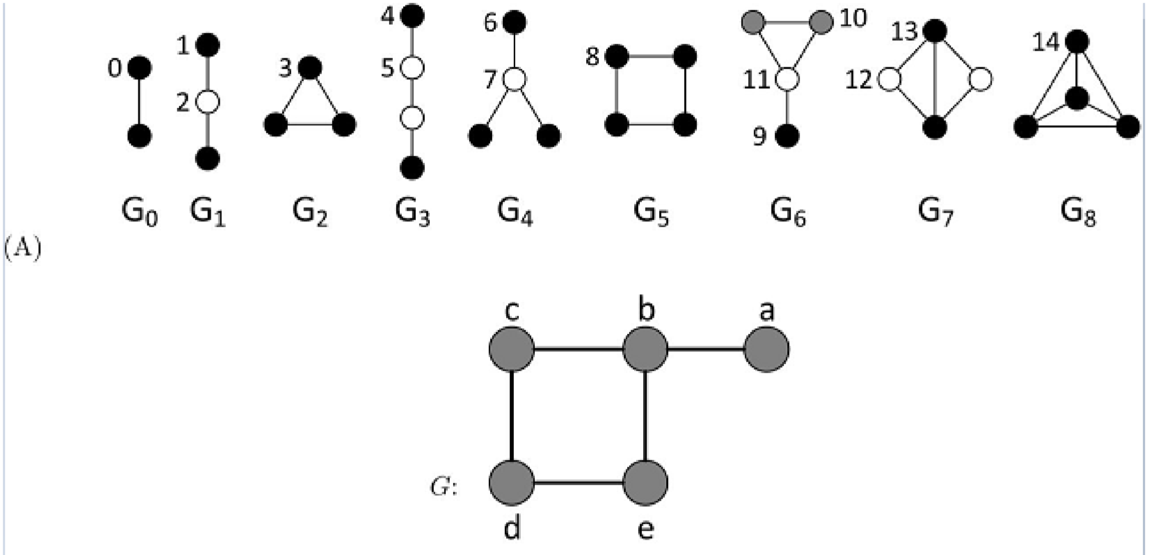
Select graphlets up to 3 nodes



Graphlet instances for node u

Node Features: Graphlet

- Graphlets are induced, non-isomorphic subgraphs that describe the structure of node u's network neighborhood.
- Example:



Graphlet	G_0	G_1		G_2	G_3		G_4		G_5	G_6			G_7		G_8
Orbit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GDV(a)	1	2	0	0	2	0	1	0	0	0	0	0	0	0	0

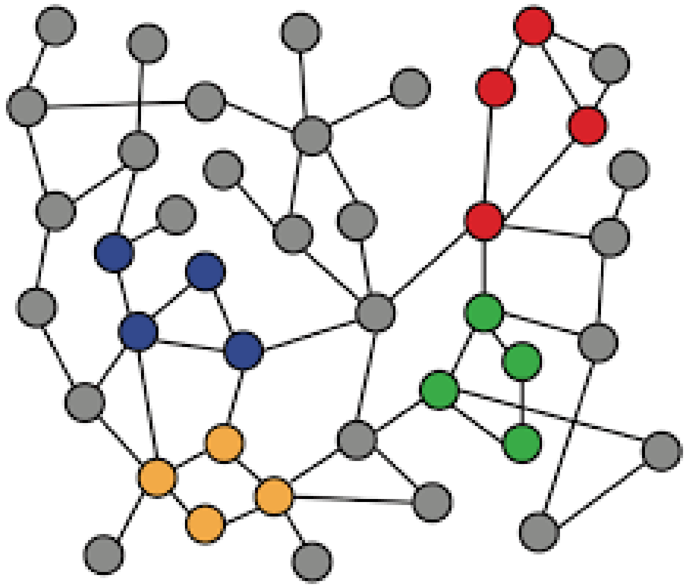
Graphlet Count(a)	1	2	0	2	1	0	0	0	0

Summary: Ways to Obtain Node Features

- Importance-based features: capture the importance of a node in a graph, useful for predicting influential nodes in a graph
  - Node degree
  - Node centrality (Eigenvector, Betweenness, Closeness Centrality)

- Structure-based features: capture topological properties of local neighborhood around a node, useful for predicting a particular role a node plays in a graph
  - Node degree
  - Clustering coefficient
  - Graphlets

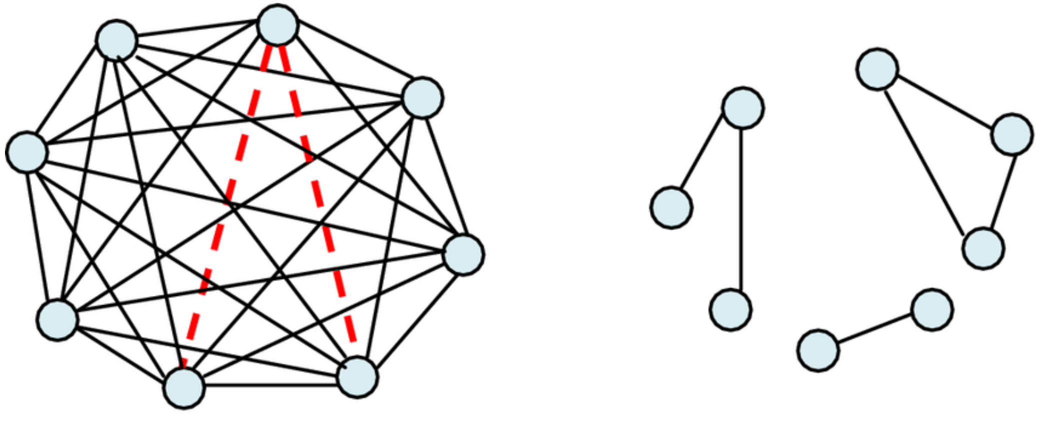
- Distance-based Features
- Local Neighborhood Overlap Feature
- Global Neighborhood Overlap Feature



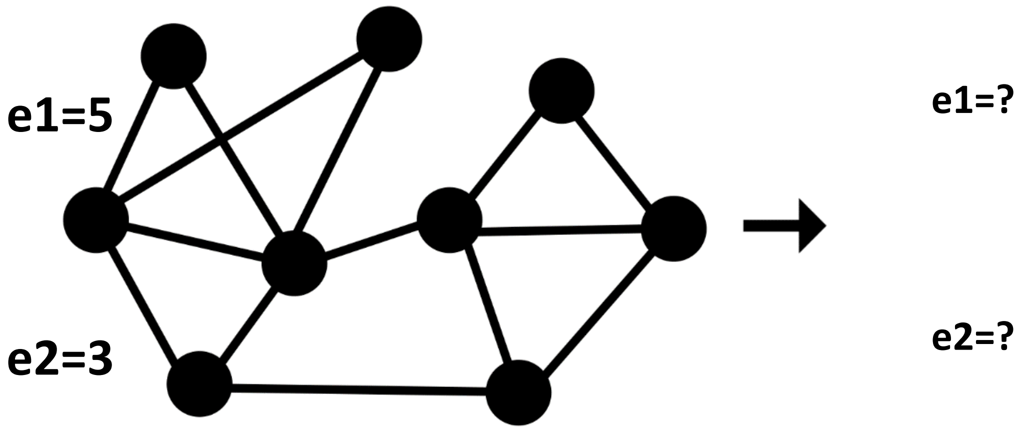
Stanford CS224W: Machine Learning with Graphs

Edge(Link)-level Prediction Task

- Predict new links based on existing links.
- Predict edge-level features/labels.



(a) Training (b) Testing

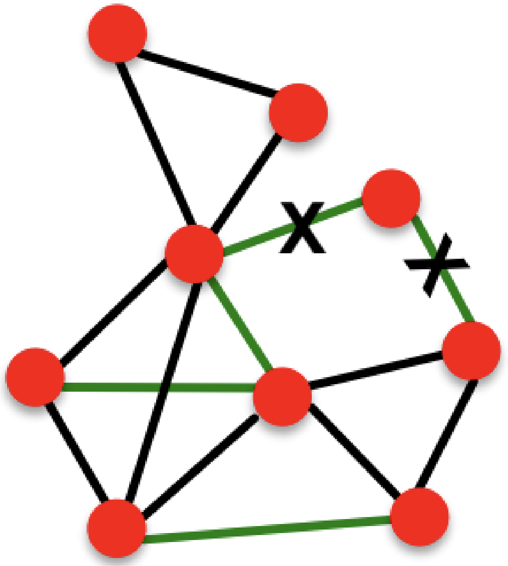


t=0 t=1

Multi-moth flame optimization for solving the link prediction problem in complex networks

Edge Prediction from Neighborhood

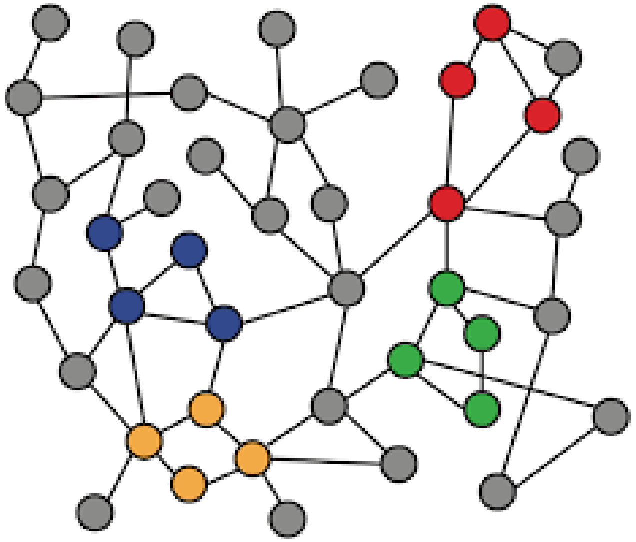
- High level idea:
  - Design features c(x, y) for each node pair (x, y)
  - For example, the number of common neighbors



- Select top k pairs as new edges
- Edge_feature(x, y, e) → new edge_feature

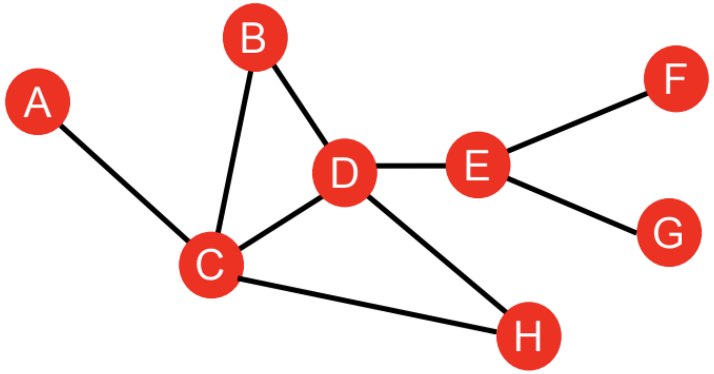
Edge-level Features: Overview

- Goal: Characterize the structure and connectivity of nodes and edges in the network:
- Distance-based features
- Local neighborhood overlap
- Global neighborhood overlap



Distance-based Features

- Shortest-path distance between two nodes
- For exmaple:

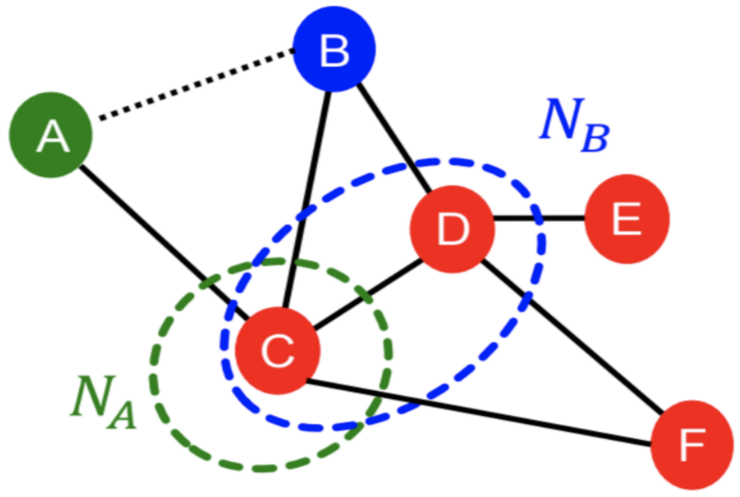


$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

- However, this does not capture the degree of neighborhood overlap:
- Node pair (B, H) has 2 shared neighboring nodes, while pairs (B, E) and (A, B) only have 1 such node.

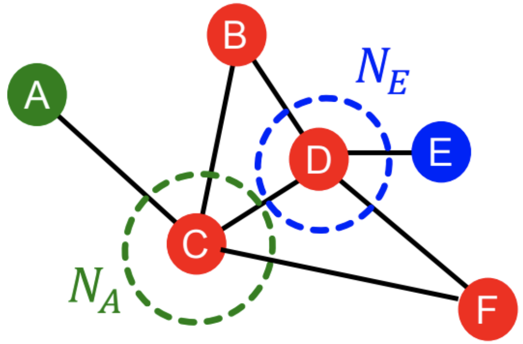
Local Neighborhood Overlap

- Captures # neighboring nodes shared between two nodes v1 and v2:
- Common neighbors: $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- Jaccard's coefficient: $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- Adamic-Adar index: $\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$
  - Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



Global Neighborhood Overlap

- Limitation of local neighborhood overlapping features:
  - Metric is always zero if the two nodes do not have any neighbors in common.
  - However, the two nodes may still potentially be connected in the future.



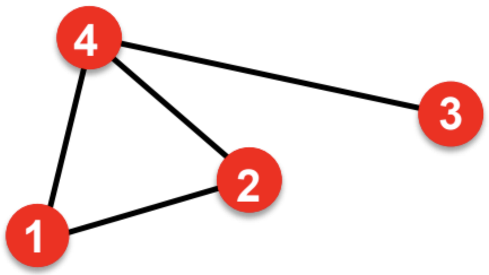
$$N_A \cap N_E = \phi$$
$$|N_A \cap N_E| = 0$$

→ Global neighborhood overlap features resolve the limitation by considering the entire graph.

Global Neighborhood Overlap: Katz Index

- Katz index:
    - Count the number of walks of all lengths between a given pair of nodes.
- How to compute number of walks?

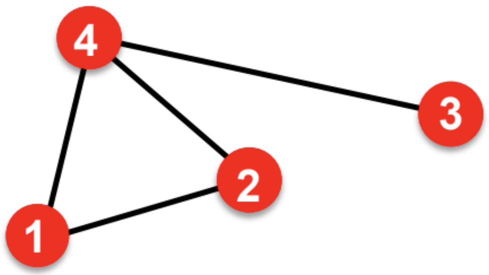
→ Powers of the graph adjacency matrix



$$A = \begin{pmatrix} 0 & \boxed{1} & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Ai,j = 1 if node i, j are connected

- So, what is number of walks?
- We want to show: $\boldsymbol{P^{(K)} = A^k}$
- where $\boldsymbol{P_{uv}^{(K)}}$ = #walks of length K between u and v
- $\boldsymbol{P_{uv}^{(1)}}$ = #walks of length 1 (direct neighborhood) between u and v = $\boldsymbol{A_{uv}}$

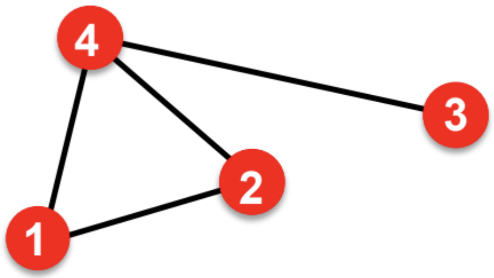


$$\boldsymbol{P_{12}^{(1)} = A_{12}}$$

$$A = \begin{pmatrix} 0 & \boxed{1} & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Ai,j = 1 if node i, j are connected

- **How to compute $P_{uv}^{(2)}$?**
  - **Step 1:** Compute **#walks** of length 1 **between each of u's neighbor and v**
  - **Step 2**: **Sum up** these #walks across u's neighbors
  - $P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^{2}$



Node **1**'s neighbors

#walks of length 1 between Node **1**'s neighbors and Node **2**

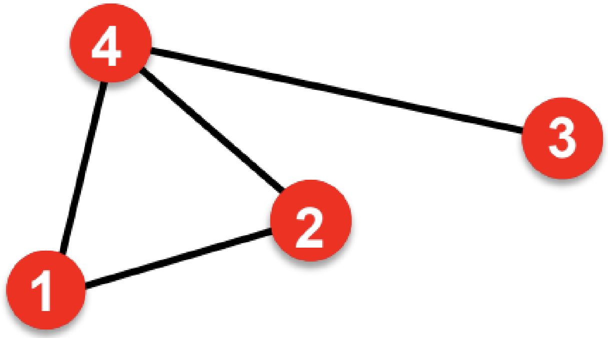
$P_{12}^{(2)} = A_{12}^{2}$

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Power of adjacency

Katz Index

- Using powers of adjacency matrix, we can calculate number of walks of all lengths between a pair of nodes.
- $\boldsymbol{A}_{uv}$ specifies #walks of length 1 (direct neighborhood) between u and v.
- $\boldsymbol{A}_{uv}^{2}$ specifies #walks of length 2 (neighbor of neighbor) between u and v.
- $\boldsymbol{A}_{uv}^{l}$ specifies #walks of length l.



- **Katz index** between v1 and v2 is calculated as **sum over all walk lengths.**

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{A^l_{v_1 v_2}}$$

#walks of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

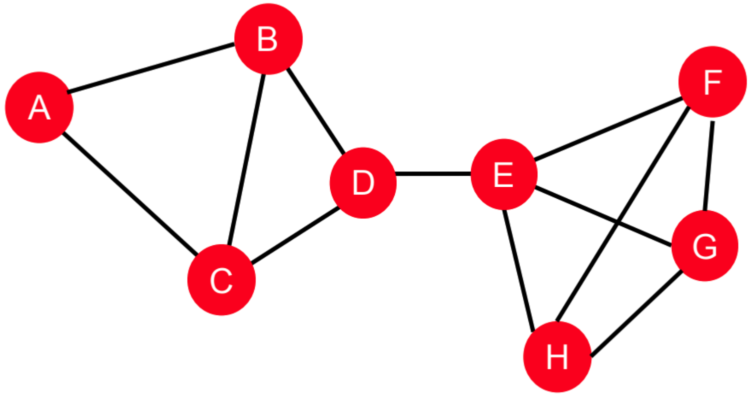
- **Katz index matrix** is computed in closed-form:

$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = (\boldsymbol{I} - \beta \boldsymbol{A})^{-1} - \boldsymbol{I}$$

Summary: Ways to Obtain Edge Features

- Distance-based features:
  - Calculates shortest path between 2 nodes, but cannot capture overlapping neighboods.
- Local neighborhood overlap:
  - Captures number of sharing neighborhoods between 2 nodes.
  - Only focuses nodes within 2-hop.
- Global neighborhood overlap:
  - Katz index uses entire graph structure to score 2 nodes.
  - It can capture the structure globally.

- Goal: We want features that characterize the structure of an entire graph.
  - Similar graphs has similar features
  - Graph Kernel Methods (kernel methods are widely-used in traditional ML for graph-level prediction.):
  - Graphlet Kernel
  - Weisfeiler-Lehman Kernel

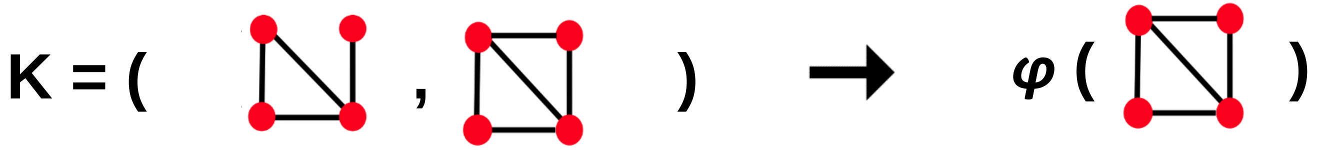


f(graph) = ?

- A quick introduction to Kernels:
  - Kernel K(G,G′) ∈ R measures similarity between two graphs (data)
  - There exists a feature representation $\phi(\cdot)$ such that:

$$K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$$

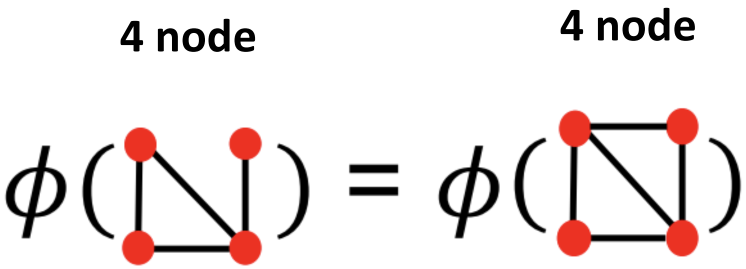
  - Kernel enables vectors to be operated in higher dimension
  - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.



네트워크 과학 연구실 NETWORK SCIENCE LAB 가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

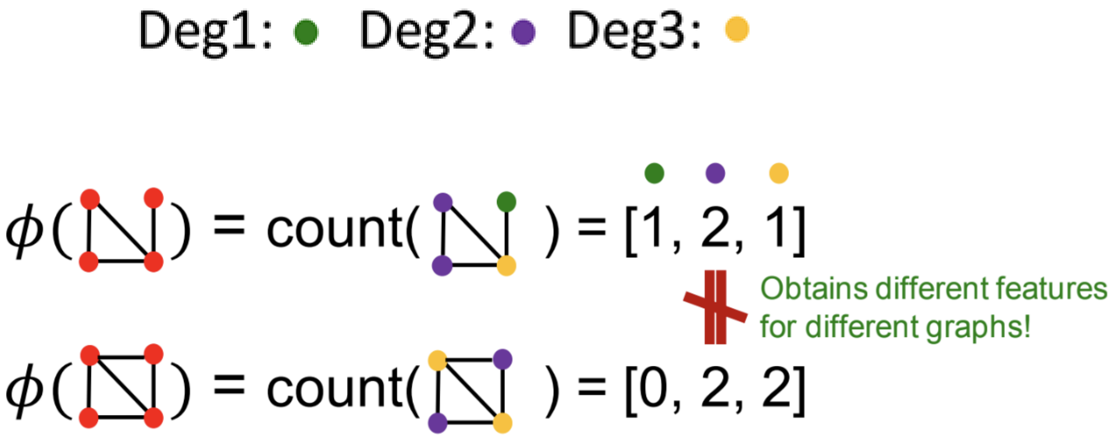
Graph Kernel: Key Idea

- Goal: Design graph kernel φ(G)
- Key Idea: Bag-of-Words (BoW) for a graph
  - In NLP, BoW counts the word's frequency in a document as feature
  - Simplest way on graph: **Regard nodes as words.**
  - For example, we found the features of 2 graphs are same.



Graph Kernel: Key Idea

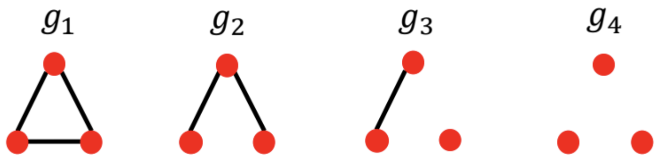
- Problem: Bag of node counts doesn't work well…
- What if we use Bag of Node Degrees?



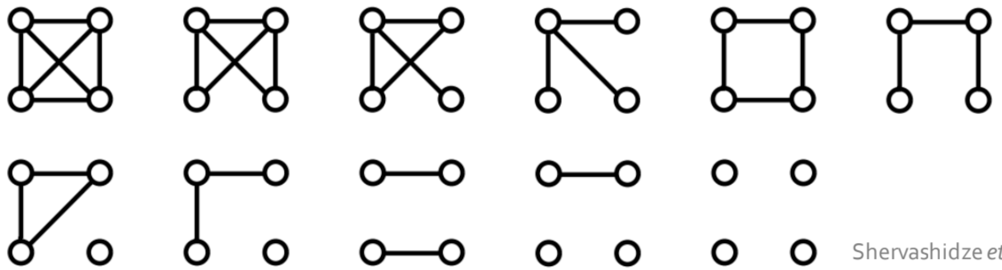
- Both Graph kernel and Weisfeiler-Lehman kernel use Bag-of-* representation of graph, but * is more sophisticated than node degrees!

- Key Idea: Count the number of different graphlets in a graph
- Graphlet Differences:
  - Nodes do not need to be connected.
  - Graphlets are not rooted.

- For $k = 3$, there are 4 graphlets.



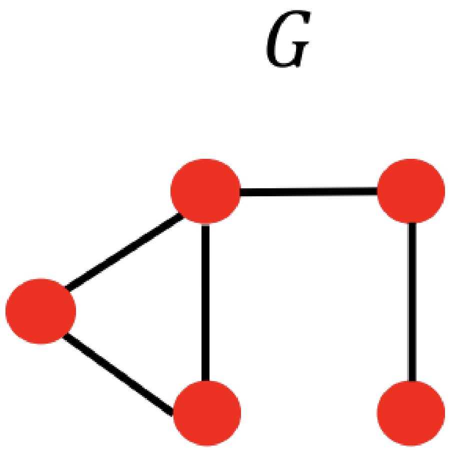
- For $k = 4$, there are 11 graphlets.

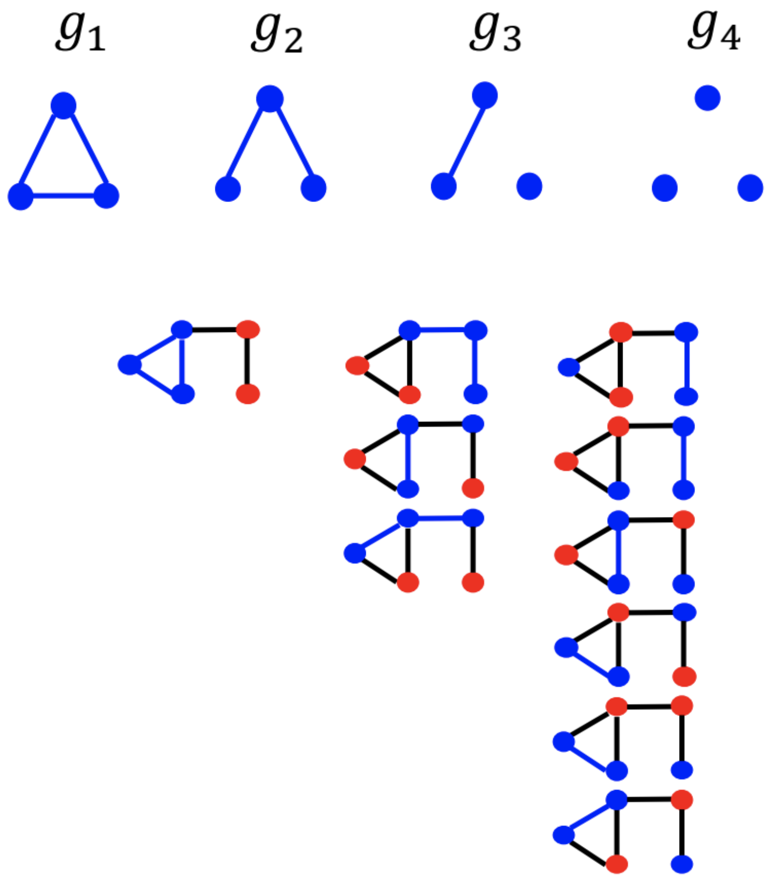


Shervashidze *et al.*, AISTATS 2011

- For example:

- For k = 3:

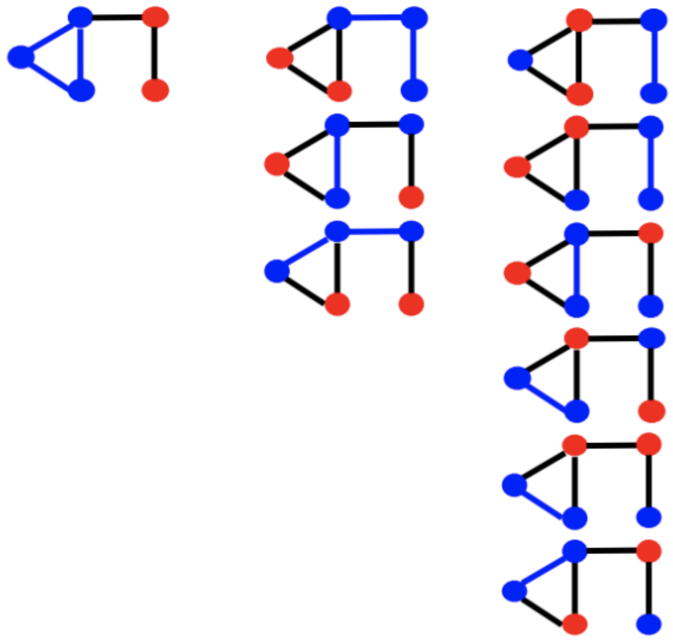




$$f_G = (1, \quad 3, \quad 6, \quad 0)^T$$

Limitations of Graphlet Kernel

- Counting graphlets is expensive!
- Counting size k graphlets for graph with n nodes by enumeration takes n^k time.
- If our graph changes with time, we have to recalculate the features again.



→ **Can we design a more efficient graph kernel?**

- Goal: Design an efficient graph feature descriptor φ(G)
- Key Idea:
  - Iteratively use neighborhood structure to describe node's neighboring topology.
  - Generalized version of Bag of node degrees (node degree only contains one-hop neighborhood information).

→ **Color Refinement Algorithm**

- For a graph G with nodes V:
- Assign initial color $c^{(0)}(v)$ each node v
- Iteratively refine node colors by:

$$c^{(k+1)}(v) = \mathrm{HASH}\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right)$$

where HASH maps different inputs to different colors.

- After K steps of iteration, c(K)(v) represents the structure of the K-hop neighborhood.

Weisfeiler-Lehman Kernel

- WL kernel benefits:
  - Computationally efficient (color refinement need #(edges) steps)
  - #(colors) depends on total number of nodes.
  - Can be used to check graph isomorphism

Summary: Graph-level Features

- Graphlet Kernel:
  - Bag of graphlets
  - Computationally expensive
- Weisfeiler-Lehman Kernel:
  - Bag of colors
  - Capture graph structure within K-hop
  - Computationally efficient
  - Closely related to graph neural networks!

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA